Week 3 – Developing a basic site with Node.js and Express

New module called express.js which does most of the web application framework for Node.js for us. To include express we do:

1. Initialize npm by npm init
2. Install express package by npm install express –save
3. Import express package for use in code by
    a. Let express = require('express');
    b. Define an instance of express by let app = express();

Routing (handler) in Express

Can use regular expressions on routing string:

- app.get("/", (req,res) => {}) means listen to default root path in server
- app.get('/about', callback) listens to requests to /about
- app.get('/ab?cd',callback) means listens to /acd or /abcd (b is optional)
- app.get('/ab+cd',callback) means listens to /abcd or more b's (b is infinite)

Usually after every GET request by the client there is a callback function which the server performs through res.send(something).

- E.g. after a user inserts a new record on a database server, the server may re-generate the new database list to the user as its callback function
- Or for a /time path request, the server may want to send the current time to the client. So below…

```
let express = require('express');
let app = express();
app.get('/', function (req, res) {
    console.log('Hello World');
    res.send('Hello World');
});
app.get('/time', function (req, res) {
    var d = new Date();
    let msg = d.getHours() + ':' + d.getMinutes() + ':' +
d.getSeconds();
    res.send(msg);
```

- Call app.listen(portnumber) at end of code to ensure there is a port for the server.
- We can create a simple database server where the sever has an array variable as a database and the client can add records by sending a query string on the path ‘newuser’ such as : *http://localhost:8080/newuser?name=Max&age=22&address=ACT*
- The sever can then retrieve the data by calling parse() on the URL, and creating a new object out of the data and pushing the object into the database, such as

```
app.get('/newuser', function (req, res) {
    let curUrl = req.url;
    let q = url.parse(curUrl, true).query;
    console.log(q);
    let newRec = {
        name: q.name,
        age: q.age,
        address: q.address
    }
    db.push(newRec);
    res.send(generateList());
});
```

Another way to handle Query Strings (in Express)

Express can handle the URL request by the client by calling req.query E.g. for *http://localhost:8080/?name=Tim&age=23&address=Mel* we can retrieve value by :

```
app.get('/', function (req, res) {
    console.log(req.query.name);
    console.log(req.query.age);
    console.log(req.query.address);
```

Route Parameters – named URL segments used to capture values specified at their position in the URL. These values are stored in the callback’s request as req.params

- for a route path : /users/:userId/books/:bookId
- Example request URL : http://localhost:3000/users/34/books/8989
- req.params: { "userId": "34", "bookId": "8989" }

Exercise: Develop a simple server-side application example that returns the result of two arithmetic operations + and -, to the client side, with requests such as :

1. *http://localhost:8080/math/add/5/2*
2. *http://localhost:8080/math/sub/3/12*

```
let express = require('express');
let app = express()
app.listen(8080);

app.get('/math/:operation/:operand1/:operand2', (req, res) => {

    if (req.params.operation == 'add')
    value = parseInt(req.params.operand1) + parseInt(req.params.operand2)

    else if (req.params.operation == 'sub')
    value = parseInt(req.params.operand1) - parseInt(req.params.operand2)

    else
    res.send("Unknown Operation")

    res.send("Output: " + value) })

app.get('/*', (req, res) => {
    res.send("Unknown Operation") })
```

Optional Router-Level Middleware – Enables Separation of Concerns (SoC) and Maintainability

We can define an express server using a router that performs our routing separately:

1. First define a router.js and call an express instance by
    - express = require('express')
2. Define a router instance by
    - router = express.Router()
3. Perform routing cases through router.get()
4. Export router by
    - module.exports = router
5. Import router in sever by
    - router = require('/router.js')
6. Use router on server's express instance by
    - app.use('/', router).

The URL Package

- Package used to process client requests in the client's current path such as taking the current URL path the client is on and processing a specific query by parsing the URL request through methods such as parse().

Week 4 – Advanced Express.js

To apply configurations on an Express.js app, we use app.set() to define a value and app.get() to retrieve a key/name value.

1. Set the port number : app.set('port', 8080)
2. Retrieve port number: app.listen(app.get('port'))

Middleware's – Can add functionality to Express Apps, these are modules that has access to the Request and Response objects that gets invoked by Express. We will learn body-parser and Morgan.

*Note: To install these we do the same as using Express: npm install --save middleware, then define an instance of it in the file by let middleware = require('middleware'

Body-parser – Parse request bodies and makes it available under the req.body property from a POST request.

- E.g. to retrieve some data from a POST request at path /data :

```
// parse application/json
app.use(bodyParser.json())

app.get('/', function (rqe, res) {
    res.sendFile(__dirname + '/index.html');
});
app.post('/data', function (req, res) {
    console.log(req.body.username);
    console.log(req.body.userage);
    res.send('Thank You')
```

Morgan – A HTTP request logger middleware for node.js

- Generates and logs automatically to any request being made
- Has a set of predefined formats such as tiny, short, common, etc.

Summary of retrieving data from a URL in Express

1. req.query – From query parameters in the URL
    - E.g. for a URL : http://foo.com/somePath?name=ted
    - req.query.name == 'ted'
2. req.params – From path segments of the URL that match a parameter in the route definition
    - E.g. for a path: /song/:songid
    - If /song/45 then req.params.songid == '45'
3. req.body – From a form post where the form data has been parsed as properties of the body object

```
<form action="/data" method="POST">
    User Name:
    <input type="text" name="username" /> </br>
    User Age:
    <input type="number" name="userage" /> </br>
    <button>Submit</button>
</form>


app.post('/data', function (req, res) {
    console.log(req.body.username);
    console.log(req.body.userage);
    res.send('Thank You')
```

Webpage Rendering

A rendering engine enables us to use static files in our application

- At runtime, the engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client
- We use EJS middleware

Each time we send an HTML file to a client, we use res.render() method.

- It renders a view and sends the rendered HTML string to the client
- On the HTML side (template), we have to use <%= %> tags to print out the value of the attribute

Example application that passes a random string and number to the client URL

```
var express = require('express');
var app = express();
app.engine('html', require('ejs').renderFile);
app.set('view engine', 'html');

app.get('/', function (req, res) {
    let randomId = Math.round(Math.random() * 100);
    res.render('index.html', { username: "admin",id:
randomId});
});
app.listen(8080);
```

```
<html>
<body>
    <h1> Welcome <%= username%> </h1>
    </br>
    <h3>You ID is<%= id%> </h3>
</body>
</html>
```

Express and Static Assets

Other than strings, we can also render files such as images, CSS files, Audio files, JS files etc by using the express.static built-in middleware function.

We define static directories which the server will retrieve files from e.g. in a directory named public we can define : app.use(express.static('public'))

- This enables us to load the files that are in the 'public' folder directory

To serve a CSS file to a HTML page, we first define a CSS file in a static directory defined by Express ; Then we call the CSS file in the HTML's header tag :

<head> <link rel = "stylesheet" href = "styles.css"> </head>

Difference between res.sendFile() and res.render()

1. The sendFile() method sends a given file to the client, regardless of the type and contents of the file
    - E.g. res.sendFile(__dirname + '/index.html') will navigate the client to the current directory's index.html file.
2. The render method works when you have a templating engine, in the lecture we used EJS. This allows us to parse a template file and generate a HTML output which utilized parsed info.
    - E.g. res.render('index.html', {username: "admin"}) will parse the following information (username object) into index.html

Exercise

Develop a server-side web application that can:

- *Render HTML files*
- *Serve static assets in a directory named 'public'*
- *Respond with a string containing the current date for any request with pathname = '/getdate'*
- *Respond with a string containing the current time for any request with pathname = '/gettime'*

```
let express = require('express')
let app = express()
app.engine('html', require('ejs').renderFile)
app.set('view engine', 'html')
app.use(express.static('public'))
app.listen(8080)

app.get('/getdate', (req, res) => {
var d = new Date()
let msg = d.getDate() + ":" + d.getMonth() + ':' + d.getFullYear()
res.send(msg)
})

app.get('/gettime', (req, res) => {
var d = new Date()
let msg = d.getHours() + ':'+ d.getMinutes() + ':' + d.getSeconds();
res.send(msg)
})
```